Protocolos de coherencia en sistemas multiprocesador

José Antonio Ortega-González
email: ortega.josant@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Abstract*—Multiprocessor systems are helpfully to improve the parallelism implementations of the programs. But one of the problems that designers found was the cache coherence. This problem could be solved by coherence protocols as MSI, MESI or MOESI in snooping. In this paper we analyze the MOESI implementation, simulating with graphic interface using Python, a multiprocessor system with four processors and four L1 caches.**

Palabras clave*—Coherencia caché, hilos, MOESI, *snooping

I. Introducción

Los sistemas multiprocesador han brindado una gran evolución en el mundo computacional. Con ellos se logran implementar procesos que requieren niveles de paralelismo, de una manera menos costosa que los procesos. No obstante, esto requiere un control más riguroso del hardware pues surgen ciertos conflictos. Uno de estos conflictos es el tratamiento de las memorias caché, donde el dato utilizado en una debe ser el mismo que en otra y que en memoria. Para esto existen protocolos de coherencia de caché que ayudan a que el tratamiento sea el más adecuado posible.

Existen varios protocolos, *MSI*, *MESI*, o *MOSI*, pero, en este documento vamos a hablar de *MOESI* así como también se va a ver su implementación. en la sección II se habla un poco del protocolo y cómo funciona. Seguidamente en la sección III se explica en qué consiste el sistema desarrollado y en la sección IV se muestran los resultados de la implementación para que finalmente en la sección V se destaquen las conclusiones del proyecto.

II. Marco teórico

Un sistema multiprocesador tiene muchos beneficios en la paralelización de procesos, aplicado al multihilo es muy eficiente en situaciones en las que se necesiten procesar programas con paralelismo. No obstante, algunas de las consideraciones que se deben tener es en el ámbito de la memoria caché y la forma en la que se administra. Al ser multiprocesador, cada procesador está aislado por lo que cada uno tiene una memoria caché para sus procesos. De esta manera, es importante aplicar protocolos de coherencia de caché para que evitar problemas de inconsistencia en los datos.

Comercialmente predominan los protocolos hardware, distribuidos en las caches del sistema, y que reaccionan en base a la observación constante del Bus [1]. La técnica de *snooping* para mantener la coherencia de la memoria caché se basa en la existencia de circuitos en cada nodo de procesamiento que monitorean permanentemente el bus para invalidar o actualizar la copia en la memoria caché local de un dato que se está modificando mediante de una operación de escritura o lectura en el bus [2]. Uno de los protocolos que existen en esta categoría es el protocolo MOESI.

A. Protocolo MOESI

Este protocolo surge como la variación y combinación de otros protocolos previos como *MSI, MESI* o *MOSI*. En este protocolo, exiten varios estados.

1) *Modified*
   Estado modificado por escritura.
2) *Owned*
   Estado compartido pero que ha sido modificado localmente.
3) *Exclusive*
   Estado exclusivo, ha sido leído desde memoria y sólo se encuentra localmente.
4) *Shared*
   Estado compartido, ha sido leído desdo otra caché.
5) *Invalid*
   Estado inválido, el dato no tiene coherencia con el valor real actualmente.

Según [3] las transiciones de estados para los bloques de caché, se pueden dar de la siguiente manera:

- Fallo de lectura (*Read Miss*).
  - Si el bloque está en *Exclusive* se envía una señal para que se pase a *Shared*.
  - Si uno o varios bloques están en *Shared*, se envía una señal para que el controlador principal también lo agregue como *Shared*.
  - Si un controlador tiene el bloque como *Modified*, este bloque pasa a *Owned*(*Probe read hit*), y se realiza la lectura correspondiente pasando el bloque externo a *Shared*.
  - Si el bloque no está en ninguna caché, se lee el bloque y se establece como *Exclusive*

- Acierto de lectura.
  Se lee el dato y no se realiza modificación.
- Fallo de escritura.
  - Si existe una copia en otra caché como *Owned* o *Modified*, ese valor se escribe en memoria, luego se escribe el nuevo valor localmente, se pasa a *Modified* y todas las demás copias pasan a *Invalid*.
  - Sino hay copia modificada, se carga y se modifica y se pasa a *Modified*, de esta manera, si otros bloques tienen copias no modificadas, estas pasan a *Invalid*.
- Acierto de escritura.
  - Si el estado estaba como *Shared*, se envía una señal a las demás copias para que estas pasen a *Invalid*.
  - Si está como *Exclusive*, entonces se modifica el bloque y se pasa a *Modified*.
  - Si está como *Modified* u *Owned* este valor se actualiza y pasa siempre a *Modified*.

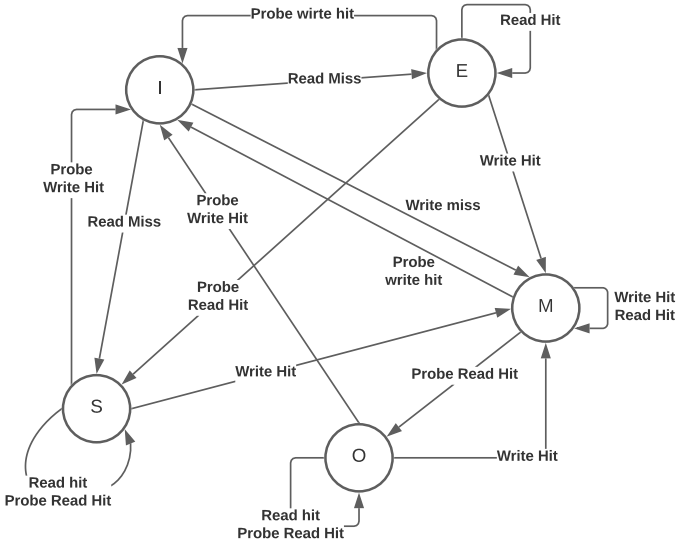

Fig. 1: Grafo de estados del protocolo MOESI.

En la figura 1 se muestra una máquina de estados en la que se resume el funcionamiento del protocolo MOESI. Este protocolo añade el estado **O** al protocolo *MESI* por lo que cuando se produce un fallo de lectura en una cache de L1, el bloque requerido se trae y se almacena con los estados **S** o **E**, dependiendo de si está o no almacenado en alguna otra cache de L1, respectivamente. Cuando se produce un fallo de escritura, el bloque se almacena en la cache con estado **M**. Ahora bien, si una cache de L1 tiene un bloque con estado **M** y se realiza un *Read Miss* en otra caché L1, su copia local pasa a tener el estado **O** (*Owned*) y se comparte como *Shared*. Esto implica que si hay bloques en estado **M** u **O**, entonces estos bloques pueden leerse o escribirse localmente sin provocar cambios de estado adicionales en la copia local. [4].

III. Sistema desarrollado

El sistema desarrollado se ha implementado en el lenguaje multiparadigma Python por su flexibilidad en cuanto a la creación de Interfaz de Usuario y el la combinación de paradigmas como el orientado a objetos y el funcional. En el diagrama de la figura 2 se muestra a grandes rasgos, la solución de implementación que se plantea para realizar la simulación.

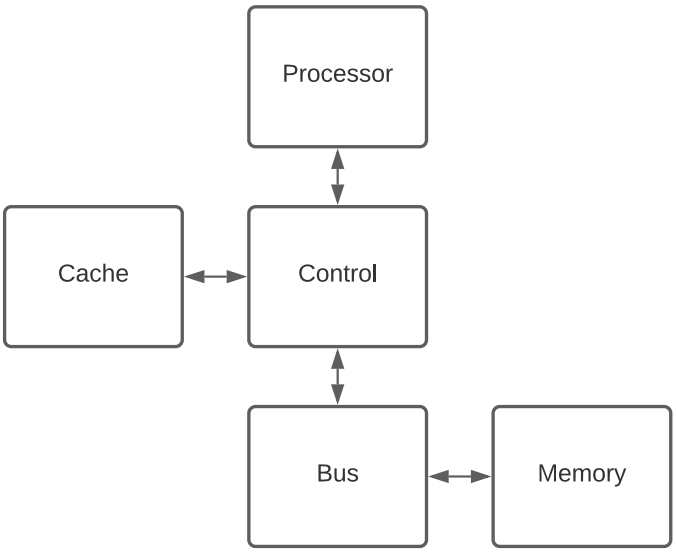

Fig. 2: Diagrama general de la solución.

Este modelo sigue el diagrama de la figura 3 que describe el funcionamiento general del sistema. De esta manera se pueden crear varios objetos que simulan los componentes de Procesador, Control y Caché con un único componente de Bus y de Memoria.

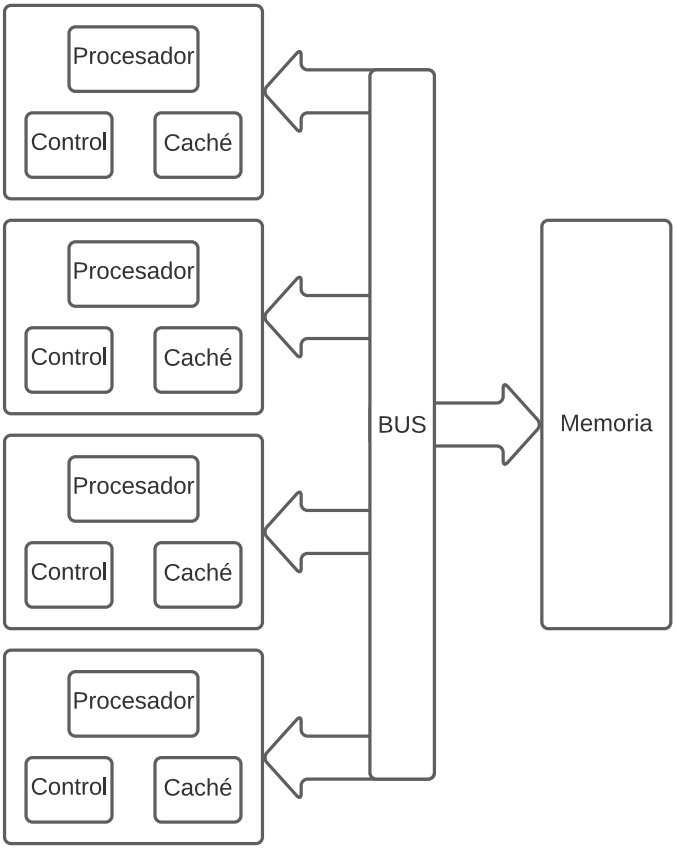

Fig. 3: Diagrama de bloques del sistema.

Finalmente, el diagrama de clases mostrado en la figura 4 es un diagrama de clases que describe las relaciones entre cada clase y se muestra también la aplicación del patrón de diseño *Singleton* para la clase Bus y la clase Memoria, que permite la existencia de un único objeto.

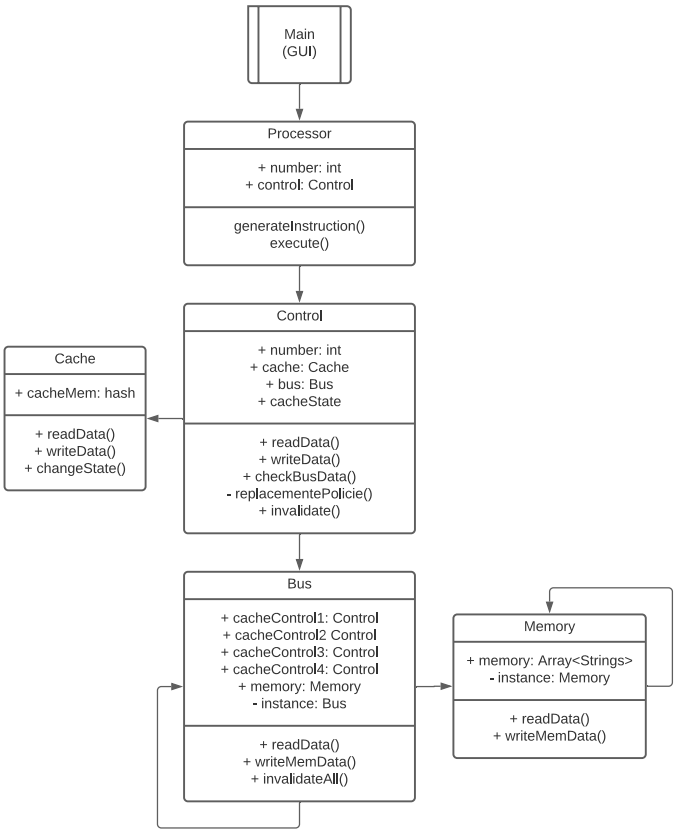

Fig. 4: Diagrama de clases del sistema.

El código de la implementación se puede encontrar en el siguiente repositorio de GitHub haciendo ingresando al **enlace**. De igual manera se puede encontrar un video explicativo de la implementación en el siguiente **enlace**.

IV. Resultados y análisis

Una vez implementado el código, el sistema se muestra como lo observado en la figura 5, inicialmente todos los elementos se encuentran en blanco, sin ningún proceso ejecutado. Para cada procesador se muestra la instrucción actual y la siguiente a ejecutar.

En la figura 6 se muestran los resultados tras una serie de ciclos ejecutados tras oprimir el botón *Inf*, en este resultado se muestran muchos estados. Por ejemplo:

- La Cache 1 tiene el primer bloque como *Owned* y la Cache 2 tiene el primero bloque como *Shared* mientras que la dirección 14 en memoria muestra otro dato. Esto siguiendo el protocolo correcto para MOESI.
- De igual manera se puede observar cómo la mayoría de bloques en todos las cachés que se encuentran como *Modified* tienen un valor distinto en memoria.
- En la Cache 4 se encuentra el bloque 3 correspondiente a la dirección de memoria 1, el estado es de *Modified* mientras que en la Cache 1 y 2 la misma dirección se encuentra como *Invalid*. Esto porque ha sufrido cambios y sus datos previos son incoherentes así como el que se encuentra en memoria.

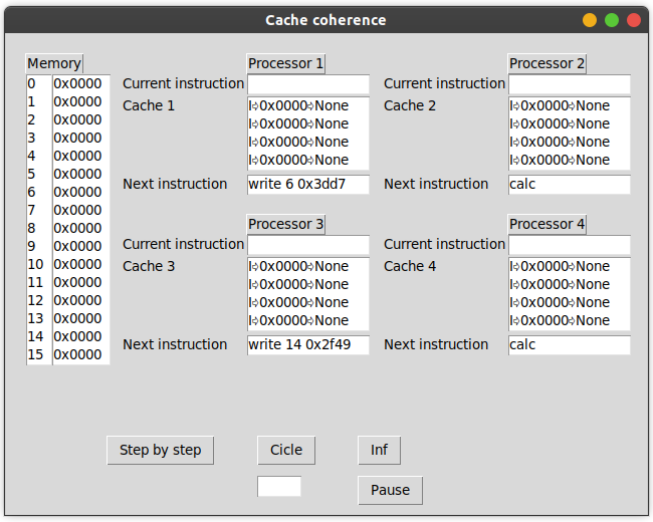

Fig. 5: Programa en inicio.

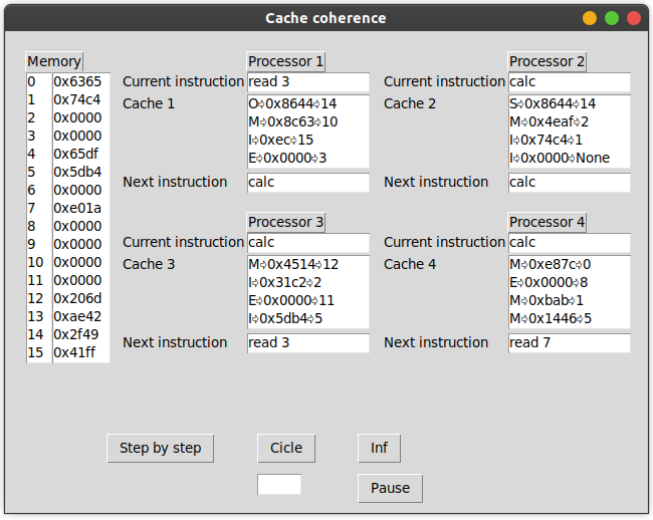

Fig. 6: Programa en inicio.

V. Conclusiones

La aplicación del sistema de monitoreo(*snooping*) mediante el protocolo MOESI brinda coherencia entre los datos que se procesan por un sistema multiprocesador. Esto ayuda a que las lecturas y escrituras sean más eficientes pues se evitan en gran medida los ingresos a memoria.

Referencias

[1] MJ Garzarán, V Viñals, JL Briz, and A Orio. Caracterización del tráfico en protocolos de coherencia snoopy.
[2] Gabriel P Silva. *Protocolos de Coerência de Cache*. PhD thesis, Universidade Federal do Rio de Janeiro.
[3] José Francés Juan Guerrero. Sistemas multiprocesadores. *Sistemas electrónicos para el tratamiento de la información*, pages 10.1 – 10.14, 2011.

[4] Francisco Candel Margaix. Caracterización del sistema de memoria de una gpgpu. 2015.